

AD-A052 023

STANFORD UNIV CALIF DIGITAL SYSTEMS LAB
EMMYXL USER'S GUIDE.(U)

F/6 9/2

UNCLASSIFIED

MAR 76 W A WALLACH
DSL-TN-84

ARO-12958.5-M

DAA026-76-6-0001
NL

|OF|

AD
A052 023



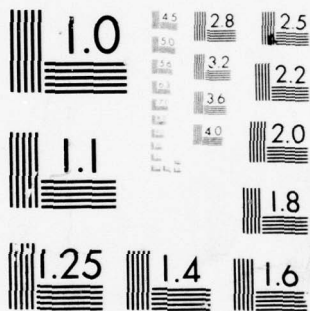
END

DATE

FILMED

5-78

DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

18 ARO 12958.5-m

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

Technical Note No. 84 12958.5-m

DSL-TN-84

4. TITLE (and Subtitle)

5. TYPE OF REPORT & PERIOD COVERED

EMMYXL USER'S GUIDE.

Technical Note

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Walter A. Wallach

8. CONTRACT OR GRANT NUMBER(s)

DAAG-26-76-G-0001

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University, Stanford, CA 9430510. PROGRAM ELEMENT, PROJECT, TASK
AREA & WORK UNIT NUMBERS

12 44p.

11. CONTROLLING OFFICE NAME AND ADDRESS

U.S. Army Research Office-Durham

12. REPORT DATE

March 1976

13. NUMBER OF PAGES

36

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING
SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution
unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an
official Department of the Army position, unless so
designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document is intended as a guide to the use of EMMYXL, the expression-
oriented line-by-line assembler developed by Hedges for the Stanford
Emulation Lab. It is intended to be used along with the Principles of
Operation for the Stanford EMMY (TN #65, Dec., 1975) and the EMMY/360
Assembler (TN #74, Dec. 1975). Various IBM OS/370 and VSII documents may
also prove useful.AD A 052023
DDC FILE COPYDDC
RECEIVED
MAR 29 1978
F

EMMYXL USER'S GUIDE

by
Walter A. Wallach

March 1976

Technical Note No. 84

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University
Stanford, California

The work herein was supported in part by the Army Research Office-Durham
under Grant No. DAAG29-76-G-0001.

Digital Systems Laboratory
Stanford Electronics Laboratories

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
DISSEMINATION	<input type="checkbox"/>
DISTRIBUTION AVAILABILITY CODES	
A	

Technical Note No. 84

March 1976

EMMYXL USER'S GUIDE

by

Walter A. Wallach

ABSTRACT

This document is intended as a guide to the use of EMMYXL, the expression-oriented line-by-line assembler developed by Hedges for the Stanford Emulation Lab. It is intended to be used along with the Principles of Operation for the Stanford EMMY (TN # 65, Dec., 1975) and the EMMY/360 Assembler (TN #74, Dec. 1975). Various IBM OS/370 and VSII documents may also prove useful.

The work herein was supported in part by the Army Research Office-Durham under contract DAAG29-76-G-0001.

Table of Contents

1.1	EMMYXL -----	1
1.2	Expressions -----	1
1.3	Multiply Step -----	3
1.4	Divide Step -----	6
1.5	Branching and Conditional -----	10
1.51	T-Machine Conditional -----	11
1.52	A-Machine Conditional -----	11
2.0	Crossassembler Operation -----	14
2.1	Titles and Comments -----	14
2.2	Location Counter Control -----	14
2.3	Listing Options -----	15
2.4	Code Generation -----	16
3.0	EMMY Simulator -----	18
	References and Related Material -----	19
	Appendix A - Error Flags -----	20
	Appendix B - Using EMMYXL -----	23
	Appendix C - Sample EMMYXL Listings -----	26
	Multiply and Divide	
	Appendix D - Sample Object Code -----	31
	and Object File Format	
	Appendix E - Sample Simulation Run -----	33
	and Instruction Trace	
	Appendix F - Control Card Summary -----	36

1.1 EMMYXL

EMMYXL is an expression oriented, line-by-line assembler language which provides a concise, straightforward expression for each possible EMMY operation. The two sides of each microinstruction are coded by separate expressions delimited by a semicolon (";"). Conditional expressions are enclosed entirely in parenthesis (that is, a CONDITIONAL T-op would be coded by enclosing the entire instruction, both T- and A-side, in parenthesis. A conditional A-op branch would be coded by enclosing only the A-side in parenthesis). See TN#74.

The syntax of the EMMYXL language is presented in a separate document (TN#74) and will not be repeated here. Some points will be clarified and discussed.

1.2 Expressions

An expression is a sequence of identifiers and symbols separated by compile time operators (double operators, such as ++ or --). Expressions are evaluated left to right. More than two terms in an expression may lead to unpredictable results. Note, since expressions are evaluated right to left, the expression:

$$8 - 3 + 4$$

is evaluated to $8-(3+4)=1$, rather than $(8-3)+4=9$.

Expressions may appear as arguments of DC and EQU pseudo ops.
See TN#74 for a more detailed discussion.

1.3 Multiply Step

Operation of the EMMY multiply step is described here. Sample code for a 32 bit multiply is included in the Appendix.

The multiply step is coded as:

MUS(AF,BF)

where AF and BF are host register identifiers. A single multiply step proceeds as follows:

- 1) The double length value obtained by concatenating the AF register contents (placed in bits <63:32>) and AF \oplus 1 contents (placed in bits <31:0>) form the PRODUCT. Thus, the AF operand specifies an even/odd register pair. Either the even or the odd register may be specified, and it becomes the high order 32 bits of the PRODUCT. The other register becomes the low order 32 bits. (e.g., if AF is specified as 3, the product is REG[3]|REG[2]). (Note: the symbol \oplus denotes EXCLUSIVE OR).
- 2) The least significant bit of the product is saved, and the product shifted right arithmetically by one bit. If OVERFLOW was set (by the previous MUS), then the sign bit is inverted.

- 3) If the least significant bit of the product was one (before the shift), then the multiplier, obtained from REG[BF], is added to the high 32 bits of the product. OVERFLOW is set or reset as needed. If no addition is performed, OVERFLOW is reset.
- 4) PRODUCT<63:32> is returned to REG[AF] and PRODUCT<31:0> returned to REG[AF \oplus 1].

Since the arithmetic shift preceeds the addition of the multiplier, an extra alignment step is necessary to properly align the 64 bit result. Thus, when multiplying two 32 bit numbers, 33 multiply steps are required, or 32 multiply steps followed by a right double arithmetic shift.

Programming Considerations

- 1) Operands may be either positive or negative two's complement values.
- 2) When doing arithmetic on quantities of fewer than 32 bits, post shifting of the result may be required.
- 3) The register specified as AF must be cleared to zero before the first MUS is executed. Register AF \oplus 1 must contain the

multiplicand. Values should not be modified between multiply steps.

- 4) After the required number of multiply steps, REG[AF] | REG[AF ⊕ 1] contain the 64 bit binary product.

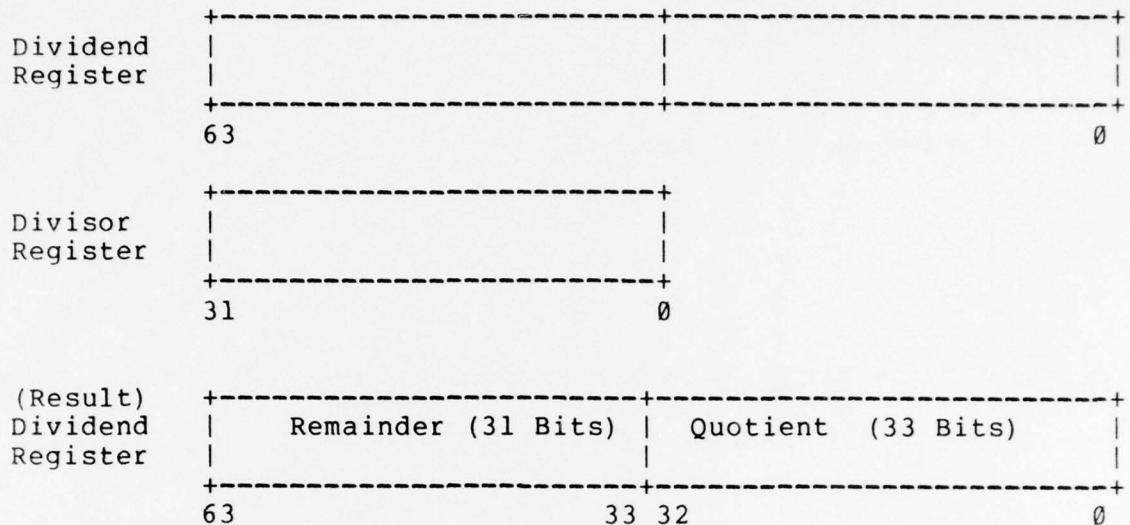
1.4 Divide Step

Operation of the EMMY Divide Step is described. Sample code for a 32 bit binary divide is included in the Appendix.

The EMMY Divide Step uses a restoring division algorithm to accomplish binary division of a 64 bit dividend by a 32 bit divisor. The result after 33 divide steps is a 33 bit quotient and a 31 bit remainder.

The algorithm proceeds in much the same way as ordinary long division. The divisor is subtracted from the high-order 32 bits of the dividend. If the result is ≥ 0 , it replaces the high-order 32 bits of the dividend. The dividend is then shifted left by one bit, shifting in a low-order one if the subtraction was successful (result replaced dividend<64:32>), or a zero if unsuccessful (dividend<64:32> unchanged). After 32 divide steps, the least significant bit of the dividend is aligned at the low end of the high 32 bits of the dividend register (bit <32:32>). One final divide step is required to calculate the remainder and least significant quotient bit. This final step shifts the remainder left by one bit however, placing the quotient most significant bit in bit <32:32> (the remainder now occupies bits <63:33> of the dividend register).

When processing very large binary numbers, the correct 33 bit quotient and 31 bit remainder result following 33 divide steps.



EMMY Divide Step Operation

Divide Step proceeds as follows:

- 1) Contents of Reg[AF]|Reg[AF ⊕ 1] form the DIVIDEND and Reg[BF] the DIVISOR.
- 2) DIVISOR is subtracted from DIVIDEND<63:32>. If the result is ≥ 0 , it replaces DIVIDEND<63:32>.
- 3) DIVIDEND is shifted LEFT LOGICALLY by one bit. If result of step 2 was ≥ 0 , a one is shifted into the low-order bit

position, otherwise if the result of step 2 was $\langle 0 \rangle$, a zero is shifted in.

- 4) $\text{DIVIDEND}\langle 63:32 \rangle$ is returned to $\text{Reg}[\text{AF}]$, and $\text{DIVIDEND}\langle 31:0 \rangle$ returned to $\text{Reg}[\text{AF} \oplus 1]$.

Programming Considerations

- 1) $\text{Reg}[\text{AF}] | \text{Reg}[\text{AF} \oplus 1]$ initially contain the 64 bit dividend, while $\text{Reg}[\text{BF}]$ contains the 32 bit divisor.
- 2) Before entering any divide steps, be sure all operand values are positive. Complement any negative values, remembering which operands were complemented.

Note: dividend must be treated as a 64 bit 2's complement binary number. Sign bit is $\text{Reg}[\text{AF}]\langle 31:31 \rangle$. It must be complemented as a 64 bit number, not two 31 bit numbers.

- 3) Divisor value must not be zero.
- 4) 33 Divide Steps are required to divide a 64 bit dividend by a 32 bit divisor.

5) After 33 Divide Steps:

Reg[AF]<0:0>|Reg[AF \oplus 1]<31:0> contains the 33 bit quotient

Reg[AF]<31:1> contains the 31 bit remainder

Since the quotient will rarely be greater than 32 bits, the high order bit can usually be ignored, and Reg[AF] shifted right logically by one bit to right align the remainder in that register.

6) Divide Step is coded as follows:

DIV(AF,BF)

7) Overflow may occur if the number of significant bits in DIVIDEND<63:32> is more than one greater than the number of significant bits in the Divisor. (Quotient requires more than 33 bits). This condition will not be detected by the hardware.

8) For other than general case operands, fewer Divide Steps and some pre- and post-shifting may be used to accomplish division.

1.5 Branching and Conditional

The EMMY Processor contains two conditionally executed instructions, one a T-op, the other an A-op. In each case, an 8-bit mask and 3 modifier bits are specified. The modifier bits are denoted V, C, and S.

The mask and modifier bits dictate a test of either the CCODES or ICODES of R0.

The mask specification is used to select a subset of the specified codes, ie CCODES or ICODES if S=0 or 1 respectively. A code bit is selected for test if its corresponding mask bit is 1. If the C bit is one, the subset of codes is inverted.

The V bit controls the sense of the test. If V=0, the test will be true if any subset bit is true, otherwise the test fails. If V=1, the test will be true if all subset bits are zero, and false otherwise.

In summary, the mask is used to select a subset of the codes specified by S. If C=1, the subset is inverted. The test is true if V=0 and any subset bit (or inverted subset bit if C=1) is one, or if V=1 and all subset bits (or inverted subset bits if C=1) are zero. Otherwise, the test fails.

1.51 T-Machine Conditional

The T-op conditional controls the execution of the A-op contained in bits $\langle 17:0 \rangle$. If the conditional test specified proves true, the A-op is suppressed; if the test proves false, the A-op is executed.

1.52 A-Machine Conditional

The A-machine conditional controls sequencing of microinstructions. A conditional test is specified as well as a 4-bit value, VAL. If the test proves true, then VAL, sign extended to 12 bits, is added to $R0\langle 11:0 \rangle$, the microinstruction counter. Foreward branches of up to 8 instructions beyond the current instruction ($R0 + 7$), or backward branches of up to 7 instructions prior to the current instruction ($R0 - 8$) are possible.

Programming Considerations

- 1) Conditions which may be detected are "any selected bit one or zero" and "all selected bits one or zero".

2) To build a conditional:

- i) set S to specify ICODES (S=1) or CCODES (S=0).
- ii) set mask to select desired code bits
- iii) for T-op conditional, if you want the A-op executed if any selected bit is one/zero, set V=1, otherwise, if you want the A-op executed if all selected bits are one/zero, set V=0. For the A-op branch, if you wish to branch if any selected bit is one/zero, set V=0, otherwise, if you wish to branch if all selected bits are one/zero, set V=1.
- iv) All mask definitions should be referred to the T-side, as described above. The assembler will make any adjustments for "NOT" conditions and A-side branches.
- v) If you wish to detect any selected

bit one or all selected bits zero,
set C=0, otherwise, if you wish to
detect any selected bit zero or all
selected bits one, set C=1.

- 2) A conditional mask is defined in EMMYXL by
EQUating a tag to a MASK function specified as follows:

`<tag> EQU MASK(<mask>,<v>,<c>,<s>)`

where <mask> is an absolute numeric value corresponding to
the desired 8 bit mask, ie 255 ==> B'11111111' ==> select all
bits. <v>,<c>,<s> specify V,C,and S bits and must be either
one or zero. A mask function should not be coded directly in
a conditional, rather EQUate a tag to the function and use
the tag.

2.0 Crossassembler Operation

This section describes the use of EMMYXL as implemented under IBM VSII. The actual assembler is currently written in ALGOLW, but will eventually be converted to PL360.

2.1 Titles and Comments

Comments are coded by placing a period (".") prior to the first character of text. This causes the assembler to ignore the rest of the input card. This text is printed with the source listing. In addition, blank lines may be included to improve the readability of source listings.

A page eject is indicated by a plus ("+") in column 1. If columns 2 through 61 are not blank, they become the new title and are printed at the top of all subsequent pages (until a new title is defined). If no new title is indicated, the previous one, if any, is printed.

2.2 Location Counter Control

All addresses processed by EMMYXL are EMMY bus addresses.

Thus, micromemory addresses are X'FF0000' through X'FF0FFF', and the host register file addresses X'FF1000' through X'FF1007'. Main store addresses are currently X'000000' through X'003FFF'. The assembler can be used to initialize any location in the EMMY system memory, including the registers and main store.

The assembler location counter can be set using the ORIGIN pseudo op. The argument of the ORG instruction must be a valid bus address or a symbol whose value is a valid bus address. All tags attached to machine instructions are assigned the current value of the location counter, which is, of course, a bus address. When coding a micromemory location as a literal, be sure to code a control store address. (eg-to begin assembly at location X'100', code ORG X'FF0100').

2.3 Listing Options

By default, the assembler will produce a source listing during PASS 2. This listing consists of a title, which contains the language processor identifier and version date, a user supplied title, and page number. The title/page eject card is discussed in section 2.1. Following the title are up to 55 lines of assembled source text. Column 1 contains an error flag, if there was any error in that source statement, or a space, followed

by the location counter value, which is an EMMY bus address. An 8 hexadecimal digit constant follows, representing the assembled object text. The card number and source text complete the line. Comments and assembler directives are printed with blank location counter and object text fields. EQU's are printed with blank location counter fields and the EQUated value in the object text field.

The source listing may be suppressed by including a "&NOLIST" card anywhere in the source stream. This card is interpreted during PASS1 and will prevent any source listing from being produced.

Portions of a source listing can be suppressed by coding "&NOPRINT" card just prior to the point at which the listing should be suppressed, and placing a "&PRINT" card just prior to the point at which the listing should resume. These cards affect only the printed listing, not the assembled object text.

2.4 Code Generation

When the assembler directive "&CODE" is included in the source text, the assembler produces an object text file consisting of a bus address and text unpacked in ASCII representation of

hexadecimal values. This object file is included at the end of the assembler listing, preceeded by the title:

***** EMMY OBJECT LISTING *****

and several ASCII control characters for use in transmitting the object text to the lab's Datapoint terminal. See Uniterm II User's Guide for details.

The default is to generate no object text file. A "&CODE" card must be included if such an output is desired.

See Appendix C for listing formats and object text formats

3.0 EMMY Simulator

The EMMYXL package includes a simulator program, whereby the EMMY code assembled can be tested. Operation of the simulator is the same as the actual EMMY.

When a "&SIM" card is included, and no errors occurred during assembly, the assembled object code is loaded into simulated control store, mainstore and host registers. Microinstructions are fetched from the location contained in R0<11:0>. The starting point for the simulation can be set by OR'ing to R0 and defining a constant equal to that address, or as an argument of the END statement.

Simulation ends when the HALT bit becomes set, or an illegal operation is attempted. At this time, a post execution dump of micromemory and the host register file is printed.

When a "&TRACE" card is included, an instruction trace is performed as the simulation progresses. Each instruction cycle, the microinstruction register and host registers are printed in hexadecimal.

Default conditions are NOSIM and NOTRACE

References and Related Material

- 1.0 Hedges, Thomas, EMMY/360 Cross Assembler, Stanford Technical Note #74, December, 1975.
- 2.0 Neuhauser, Charles, Dynamic Microprogrammable Processor (Version III), Stanford Technical Note #65, December, 1975.
- 3.0 Datapoint Corporation, Uniterm2/Uniterm3, 3.1 User's Guide, June 27, 1974, Datapoint Corporation, 9725 Datapoint Dr, San Antonio, Texas 78284.

Appendix A Error Flags

- A Illegal A-statement
 - a) missing ")"
 - b) missing ";"
 - c) illegal syntax or operator (eg, missing "=")
 - d) missing expression or illegal "-"

- B "BLK" statement error
 - a) <abs> not > 0

- C Illegal constant
 - a) Hex constant specifies other than 0-9, A-F
 - b) Octal constant specifies other than 0-7
 - c) Binary constant specifies other than 0-1
 - d) illegal DC

- D
 - a) illegal conditional
 - b) address not > 0 in "ORG"

- E
 - a) illegal END card
 - b) illegal expression

- I Insert/Extract error
 - a) specified field(s) can not be assembled into a legal
 literal mask (see L flag)

- K illegal "MASK" function
- a) missing "("
 - b) <cmask> or <bmask> out of range (not >=0,<=255)
 - c) V,C,S not 0 or 1
 - d) values not seperated by comma (",")
- L illegal literal
- a) literal does not specify 16 bit constant zero or one
 filled on right or left to 32 bits in expanded B-field
 -) literal out of range (not >= 0, <=4095 in LOAD IMMEDIATE)
- M syntax error
- a) composite operator illegally written (eg "!=" written
 as " : =")
- P procedural error
- a) literal out of range (lit not =>-8,<=7)
 in branch, pointer mod
 - b) illegal conditinal in pointer mod
- Q undetermined error
- R relocatability error

S PASS2 LC not = PASS1 LC - internal error has occurred

T illegal T-statement

- a) illegal syntax
- b) illegal operator
- c) illegal operand

U undefined symbol

X illegal pointer mod after indirect access

- a) other than CF to DF specified
- b) illegal syntax (ie not "+" or "-")

Appendix B Using EMMYXL

All necessary JCL to use EMMYXL has been included in the cataloged procedure EMMYXL. The input specified must be a Wylbur EDIT format data set containing all text and control cards. This data set must be cataloged. If the name contains qualifiers (or indicies, in SCIP terminology), that is, qualifying name(s) separated by period(s), it must be enclosed in single quotes in the EXEC statement. Additionally, no qualifier may be longer than 8 characters, nor begin with other than an alphabetic character (the same restrictions apply to simple names).

A GROUP and USER must be specified, reflecting the account under which the source data set is stored. The source data set must be cataloged

If the source text contains both upper and lower case characters, an uplow listing may be obtained by coding ",SYSOT=D" in the EXEC statement (note the misspelling of SYSOUT). Only the assembler listing will be printed uplow, so watch for two listings.

examples:

```
// EXEC EMMYXL,SOURCE=myfile,GROUP=gg,USER=uuu
```

file accessed is 'WYL.gg.uu.myfile'. Listing is upper case only.

```
// EXEC EMMYXL,SOURCE=myprog.versl,USER=uuu,GROUP=gg
```

illegal - SOURCE is a qualified name (contains special character, ".") and is not enclosed in single quotes.

```
// EXEC EMMYXL,SOURCE=360emu,GROUP=gg,USER=uuu,SYSOT=d
```

illegal - SOURCE contains qualifier which begins with numeric.

```
// EXEC EMMYXL,SOURCE='text.versl',GROUP=gg,USER=uuu
```

file accessed is 'WYL.gg.uuu.text.versl', Listing is upper case only.

The EMMYXL procedure executes two job steps. The first UNPRESSES the EDIT format source and creates a card-image scratch data set, which is PASSED to the second step.

The second job step reads the scratch data set as input to the assembler. This data set is scratched at step termination.

The original source, however, is kept.

Appendix C: SAMPLE EMMYXL LISTINGS

[illegible]

BEST AVAILABLE COPY

1 3

PAGE 2.000
3.000
4.000
5.000
6.000
7.000
8.000
9.000
10.000
11.000
12.000
13.000
14.000
15.000
16.000
17.000
18.000
19.000
20.000
21.000
22.000
23.000
24.000
25.000
26.000
27.000
28.000

EMMYXL -EXAMPLE 1 FIXED POINT MULTIPLY

2

6

ECODE
ESIM
ETRADE

7

REGIATER DEFINITIONS

00C00000
C0000002
00C00004
00C00005
00000007

MAR EQU R0
XR EQU R2
P EQU R4
Q EQU R5
S EQU R7

.DIVIDEND HIGH HALF
.DIVIDEND LOW HALF
.DIVISOR

.P.Q ALSO FORM THE PRODUCT REGISTER (64 BITS)
.S IS ALSO THE MULTIPLICAND REGISTER, Q THE MULTIPLIER

ORG X'FF0000' .START AT LOCATION 0

MULT:

.FIXED POINT MULTIPLY 32 BIT OPERANDS

.OPERANDS ASSUMED IN REGISTERS Q (MULTIPLIER) AND S (MULTIPL*ND)

P := 0 ; XR = 32 .CLEAR PRODUCT<63:32> SET ITERATION CTR

MUS(P,S) ; DEC XR (~<0 => MAR-1) .33 ITERATIONS

FF0000 0C13A020

FF0001 6FF2208F

FF0000

3

4

27

1 Language Processor Identifier and Version Date

2 User Title

3 Flag (none in this assembly)

4 Location (bus address)

5 Object Code (or value, in the case of EQU, Mask)

6 Input Card Number

7 Source Text

```

30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80

```

```

00000011
00000009
6001F018
6001C019
6001D01A
0AFC6001
1R008000
C283800A
19020001
00FE7000
01908845
19020003
00866060
00928060
32140001
3310C000
14080020
6CF2208F
5210C001
C0478015
C0524000
C0278017
008650J0
180C0000
C0CC0000
FFFFF000
FFFFF02A

```

```

      .THE DIVIDE ALGORITHM WILL WORK WITH POSITIVE VALUES ONLY.
      .THEREFORE, WE MUST COMPLEMENT NEGATIVE VALUES AND KEEP TRACK OF
      .WHICH VALUES WERE COMPLEMENTED. THE REMAINDER MUST BE COMPLEMENTED
      .IF THE DIVIDEND WAS NEGATIVE, AND THE QUOTIENT COMPLEMENTED IF
      .EITHER THE DIVIDEND OR DIVISOR, BUT NOT BOTH, WERE NEGATIVE.
      .ICCODE BIT <1:1> WILL INDICATE REMAINDER TO BE COMPLEMENTED, AND
      .ICCODE BIT <0:0> INDICATE QUOTIENT TO BE COMPLEMENTED.
      CCPLREM EQU MASK(2,0,0,1) .FLAG FOR REMAINDER
      NEG EQU MASK(1,0,0,1) .FLAG FOR QUOTIENT
      : S = M(DIVISOR)
      : P = M(DIVID1)
      : Q = M(DIVID2)
      S := S : ( NOT ZERU => MAR+1 ) .IS DIVISOR ZERO?
      MAR := MAR OR X'00008000' .YES - HALT
      ( POSITIVE => : MAR = * ++ 3 )
      MAR := MAR XOR X'0000J000'
      S := ~S : INC S
      P := P : (NOT NEGATIVE => MAR+5) .IS DIVIDEND NEG?
      MAR := MAR XOR X'00003000' .YES INVERT QUOTIENT FLAG
      Q := ~Q .SET REMAINDER FLAG
      P := ~P
      Q := Q+1
      P := P +C+ 0
      XR := 32
      DIV(P,S) : DEC XR (~Q => MAR-1) .33 ITERATIONS
      P >> 1 .TRUNCATE QUOTIENT TO 32 BITS
      .AND LEAVE REMAINDER RIGHT JUST
      ( COMPLREM => : MAR=**+2) .MUST WE COMPLEMENT REM?
      P := ~P : INC P .YES - IGNORE OFL
      ( NEG := : MAR=**+2) .HOW ABOUT QUOTIENT?
      Q := ~Q : INC Q .YES - IGNORE OFL
      .NOW Q CONTAINS QUOTIENT AND P CONTAINS REMAINDER
      MAR := MAR OR X'00008000' .HALT
      DIVISOR: DC 13
      DIVID1: DC --1
      DIVID2: DC --214

```

BEST AVAILABLE COPY

PAGE 3

EMMY X L	3/28/76	HOST REGISTER INITIALIZATION	
		.NOW INITIALIZE HOST REGISTERS	
FF1000		JRG X'FF1000'	82
FF1001	00000000	DC	83
FF1002	00000003	BLK 3	84
FF1003	00000000	DC	85
FF1004	00C0002D	DC	86
FF1005	000C0000	DC	87
FF1006	0000012A	DC	88
FF1007		DC	89
C00000		END	90
			91

① X'00000000' .RO START AT LOCATION 0
 .UNUSED REGISTERS
 .Q-REG - MULTIPLIER
 .R-REG NOT USED
 .S-REG MULTIPLICAND

1 Reset Location Counter to Host Register File

BEST AVAILABLE COPY

SYMBOL TABLE LISTING

EMMY X L 3/28/76

BUSY	MASK	0000000C
CARRY	MASK	00000104
COMPLREM	MASK	00000011
CIVID1	SYMB	00FF0019
CIVID2	SYMB	00FF001A
DIVISOR	SYMB	00FF0018
HIGH	MASK	00000094
LOW	MASK	00000044
MAR	REG	00000000
MULT	SYMB	00FF0000
NEGATIVE	MASK	00000094
NCT	MASK	00000009
ODD	MASK	00000014
OVERFLOW	MASK	00000002
P	REG	00000004
POSITIVE	MASK	000000A0
C	REG	00000005
R0	REG	00000000
R1	REG	00000001
R2	REG	00000002
R3	REG	00000003
P4	REG	00000004
R5	REG	00000005
R6	REG	00000006
R7	REG	00000007
S	REG	00000007
SAME	MASK	00000024
XR	REG	00000002
ZERO	MASK	00000000

②

③

①

EMMY Symbol Table

1	Tag or Name	2	Identifier Type	3	value (in case of EQU, MASK, etc) or location (in case of Tag)

Appendix D Sample Object Code and Object File Format

EMMYXL will produce an object text file at the end of the source listing if the assembler directive "&CODE" was included in the source. This file consists of EMMY bus addresses and object text, unpacked into ASCII representation of hexadecimal values. Thus, the hexadecimal value X'9C' would appear as two ASCII characters "9" and "C" printed as part of a string of such characters. These must be packed and converted to their binary equivalents before loading into the EMMY memory system. A loader program is incorporated in the console debug program which accepts this object format, performs the necessary conversions, and loads the text into the EMMY system.

Object text records consist of a six character address, preceded by an address identifier character ("@"). This is followed by up to 64 characters of object text. Each object text record contains an address.

The object text is preceded by two control characters, which are used to control the tape unit of the Datapoint 2200. The Ascii "ENQ" is interpreted by the Uniterm (r) program as "enable cassette" (X'2D'), and enables data to be written to the front cassette unit. The second control character, "DC-2" or "TAPE-ON" (X'12') actually begins writing data to the cassette.

Following the object text appear two control characters which are the complement of the first two. A "DC-4" or "TAPE-OFF" stops writing data to the cassette and writes the last record (purges the Datapoint buffer), and an "EOT" (X'37') disables the cassette unit.

Object File Format

"ENQ" (X'2D')
"TAPE-ON" (X'12')
any number of object text records
"TAPE-OFF" (X'3C')
"EOT" (X'37')

This follows the title "***** EMMY OBJECT CODE *****" in the source listing.

BEST AVAILABLE COPY

```
***** EMMY OBJECT CODE *****
<Control Characters (unprintable)>
aFF0000 0C13A020CFF220BF6D01F0186001C01950001001A0AFC000118008000C283800A
aFF0008 190200C100FE7000A900E45190200030086806000928060321+000133100000
aFF0010 1A380020CFF220BF52100001C04780150092+000C02750170036500018008000
aFF0018 0000000000000000000000000000000000000000000000000000000000000000
aFF1000 0000000000000000000000000000000000000000000000000000000000000000
<Control Characters (unprintable)>
*** END OF ASSEMBLY ***
000.67 SECONDS ASSEMBLY TIME
NO STATEMENTS FLAGGED
```


IR=	6CF220BF	R0	82030012	P1	000C0003	K2	00000012	K3	00000000	R4	00000000	R5	00358000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	R1	000C0003	K2	00000011	K3	00000000	R4	00000000	R5	00680000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	R1	000C0003	K2	00000010	K3	00000000	R4	00000000	R5	00680000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	F1	000C0003	K2	0000000F	R3	00000000	R4	00000000	R5	01AC0000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	F1	000C0003	K2	0000000E	R3	00000000	R4	00000000	R5	03560000	R6	00000000	R7	00000000
IR=	6CF220BF	P0	82030012	R1	000C0003	K2	00000000	R3	00000000	R4	00000000	R5	06800000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	F1	000C0003	K2	0000000C	K3	00000000	R4	00000000	R5	00600000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	R1	000C0003	K2	0000000B	K3	00000000	R4	00000000	R5	1AC00000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	R1	003C0003	K2	0000000A	R3	00000000	R4	00000000	R5	35800000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	K1	000C0003	K2	00000000	R3	00000000	R4	00000000	R5	06000000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	R1	000C0003	K2	00000000	R3	00000000	R4	00000000	R5	06000000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	P1	000C0003	K2	00000007	R3	00000000	R4	00000001	R5	AC000000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	R1	000C0003	K2	00000006	R3	00000000	R4	00000000	R5	58000000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	R1	000C0003	K2	00000005	K3	00000000	R4	00000006	R5	80000000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	R1	000C0003	K2	00000004	K3	00000000	R4	00000000	R5	60000000	R6	00000000	R7	00000000
IR=	6CF220BF	R0	82030012	R1	000C0003	K2	00000000	K3	00000000	R4	00000000	R5	C0000031	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	R1	000C0003	K2	00000002	R3	00000000	R4	00000001	R5	80000000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	R1	000C0003	K2	00000001	K3	00000000	R4	00000003	R5	00000000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	R1	000C0003	K2	00000000	R3	00000000	R4	00000006	R5	00000000	R6	00000000	R7	00000000
IR=	5CF220BF	R0	82030012	R1	000C0003	K2	00000000	R3	00000000	R4	00000000	R5	00000010	R6	00000000	R7	00000000
IR=	52100000	R0	82030013	R1	000C0003	K2	FFFFFFFFFF	R3	00000000	R4	00000000	R5	00000000	R6	00000000	R7	00000000
IR=	C0478015	R0	82030014	R1	000C0003	K2	FFFFFFFFFF	R3	00000000	R4	00000006	R5	00000000	R6	00000000	R7	00000000
IR=	00524000	R0	82030015	R1	000C0003	K2	FFFFFFFFFF	R3	00000000	R4	00000006	R5	00000000	R6	00000000	R7	00000000
IR=	C0278017	P0	56030016	R1	000C0003	K2	FFFFFFFFFF	R3	00000000	R4	FFFFFFFFFF	R5					

NORMAL END CF SIMULATION

000.10 SECONDS SIMULATION TIME

34

- 1 33 Multiply Steps
- 2 Result = 'X'3462' = 13,410 = 45*298
- 3,4 Load values for Division
- 5 Complement Dividend and set flag in R0
- 6 33 Divide Steps
- 7 Complement Quotient and Remainder answer = -16 remainder = -6

BEST AVAILABLE COPY

*** EMMY MEMORY DUMP ***

R0= 8003801E P1= 00000003 R2= FFFFFFFF R3= 00000000 R4= FFFFFFFF R5= FFFFFFFF R6= 00000000 R7= 00000000

000000	0C13A020	6FF2208F	6DC1F018	6D01C019	6D01D01A	0AFC6001	18008000	C283800A	OR ?M F M a M P I	B
000008	1902C001	00FE7000	0A500845	19020003	00B68060	00928060	32140001	33100000	-P	3
000010	14C60020	6CF2208F	52100001	C0478015	00924000	C0278017	00865000	18008000	LR ?R	2
000018	00000000	FFFFFFF	FFFFFFF2A	F8F8F8F8	F8F8F8F8	F8F8F8F8	F8F8F8F8	F8F8F8F8	a a *	6P

*** END OF DUMP ***

000.78 SECONDS IN EXECUTION

Appendix F: CONTROL CARD SUMMARY

- <text> - period followed by text - comments
 ignored by EMMYXL
- + [<text>] - plus sign in column 1 - page eject
 text (if any) replaces user title at head of page
- ORG {<tag>
 <abs>} - set location counter - <abs> must be EMMY bus address
- &NOLIST - suppress entire source listing
- &NOPRINT - turn off printing of that portion of source
 following the &NOPRINT card
- &PRINT - resume printing of all source text
- &CODE - produce EMMY object code
- &SIM - simulate assembled EMMY program if no errors
 occurred during assembly
- &TRACE - produce instruction trace during simulation

Note: all "&" control cards and title card must begin in
column 1.

IED
78

